Malware Detection & Memory Forensics

Yue Duan

Malware Detection - Intro

1. **Malware Detection techniques**
   - static analysis
   - dynamic analysis
2. **Detection Evasion techniques**
3. **Paper presentation**

BareCloud: Bare-metal Analysis-based Evasive Malware Detection, Usenix Security'14

Malware Detection - Detection

Static analysis

Definition: *testing and evaluation of an application by examining the code without executing the application*

- Pros: good code coverage
- Cons: code obfuscation, encryption, false positives

Malware Detection - Detection

Dynamic analysis

(in-guest monitoring, VMI)

Definition: *testing and evaluation of an application during runtime*

- Pros: capture behaviors accurately
- Cons: poor code coverage

Malware Detection - Evasion

Virtual machine environment is different from real machine. Those differences could be used to detect VM and evade analysis.

- CPU instruction semantics
- Timing attacks
- VM bugs
- Username, Windows Product ID

Paper presentation

BareCloud: Bare-metal Analysis-based Evasive Malware Detection

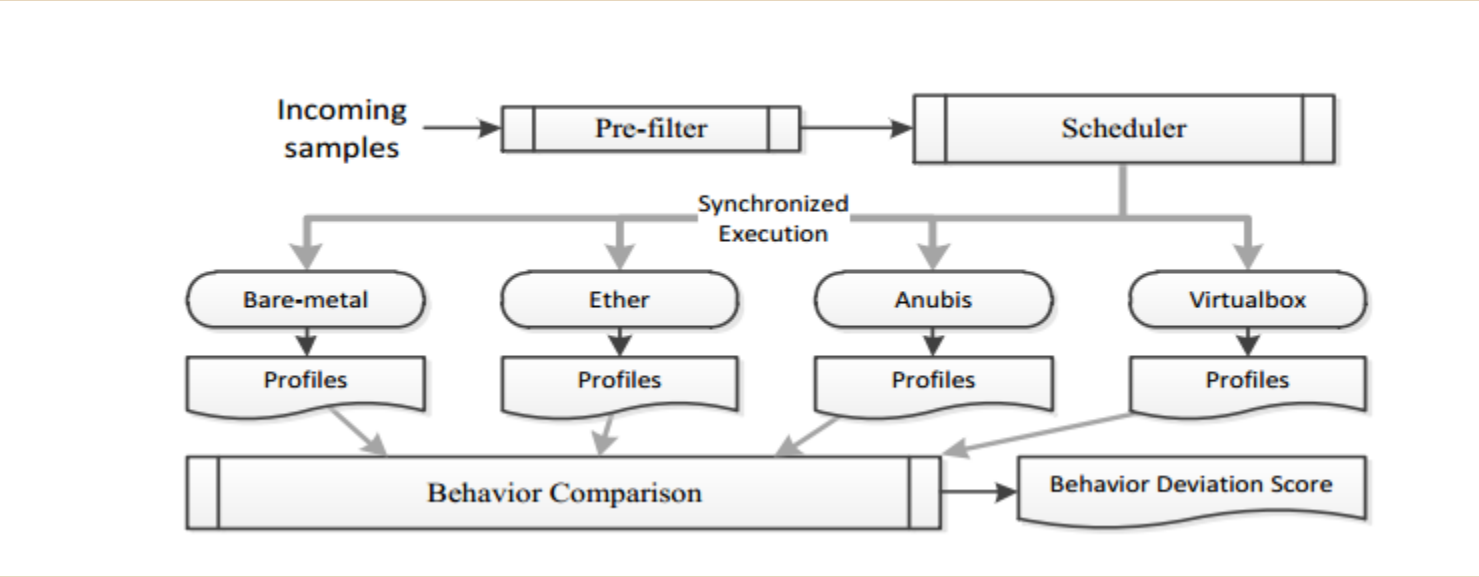
Dhilung Kirat, Giovanni Vigna, Christopher Kruegel

UC Santa Barbara

USENIX Security 2014

Motivation: Can we automatically identify evasive malware while preserving transparency?

BareCloud - System Overview



1. Prescreen; 2. Execution; 3. Behavior Extraction; 4. Behavior Comparison

BareCloud - Prescreen

The purpose of the pre-screening process is to select more interesting samples that are likely to have environment-sensitive behavior.

Simply use Anubis platform for prescreening.

BareCloud - Execution

A scheduler is implemented to run the malware sample on four platforms simultaneously to minimize the impact of external environments.

1. Bare-metal
2. Anubis (Emulator)
3. Ether (intel VT)
4. Virtualbox (Type 2 hypervisor)

BareCloud - Execution

malware initiator: component that starts the execution of the malware.

To make it transparent, the malware initiator removes itself and all of its artifacts after initiating the malware.

BareCloud - Behavior Extraction

Two common ways for behavior extraction:

1. VMI based approach
2. In-guest monitoring

However, these two approaches do not fit in the bare-metal scenario. Because these two approaches are not transparent enough.

BareCloud - Behavior Extraction

As a result, BareCloud extracts file system behaviors and network behaviors.

1. File system behavior: compare the disk contents from before and after the malware execution.
2. Network behavior: use an external traffic capture component.

BareCloud - Behavior Extraction

- Extract file metadata information from raw disk image before and after the execution of malware.

  (delete, create, modify)
- Extract registry keys metadata from raw registry hive.
- Process file system and registry behavior to identify critical modification of system

BareCloud - Behavior Normalization

Behavioral profile extracted contains both malware behavior as well as the background operating system behavior. We need to filter out the normal behavior.

Use a “void” program that does nothing other than stall infinitely to extract background behavior.

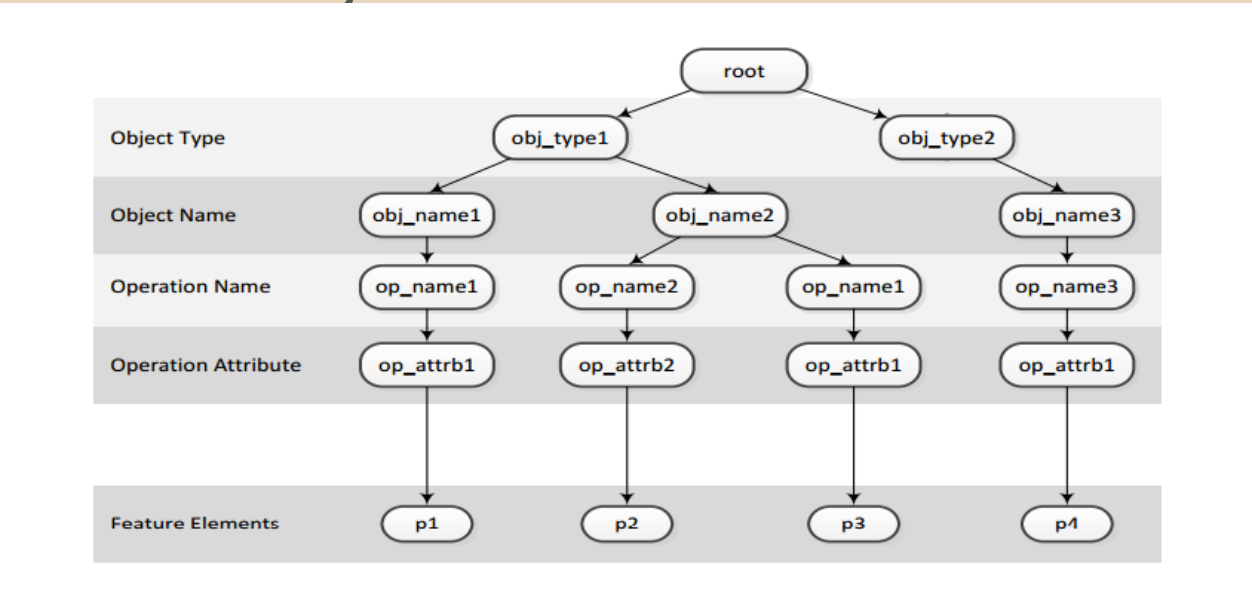
BareCloud - Behavior Comparison

Malicious behavior only occupies a small portion of the whole behavior profile.

Traditional Jaccard similarity will produce very high similarity score even if the system could detect some evasive behaviors.

BareCloud - Behavior Comparison

Compare the behavior profiles in a hierarchical way



BareCloud - Evaluation

110,000+ malware samples are collected and analyzed. 5835 evasive malware samples are found.

Environment	Detection count	Percentage
Anubis	4,947	84.78
Ether	4,562	78.18
VirtualBox	3,576	61.28
All	2,530	43.35
Total	5,835	

Memory Forensics

1. Memory Forensics Intro
2. Paper presentation

DSCRETE: Automatic Rendering of Forensic Information from Memory Images via Application Logic Reuse

Usenix Security’14

Memory Forensics

Per Wikipedia, Memory forensics is forensic analysis of a computer's memory dump.

A series of existing research has been made to reverse engineer the data structures reside within the memory dump, including Volatility, MACE, Siggraph, etc.

Memory Forensics

Now, a pending issue has raised up. Though we can perform analysis to recover the data structures, we still have no idea what kind of content is actually inside the structures.

For example, a buffer might hold a PDF file content or a JPEG photo.

Memory Forensics - paper

DSCRETE: Automatic Rendering of Forensic Information from Memory Images via Application Logic Reuse

Brendan Saltaformaggio, Zhongshu Gu, Xiangyu Zhang, and Dongyan Xu

Usenix Security'14

Motivation: Can we recover memory content via logic reuse?

DSCRETE - Observations

1. Application that defined the data structure also contains printing/rendering logic for it!

   Let’s call the rendering logic - P function. It transforms data structure to formatted application output.

2. Given incorrect input, P function will crash!

DSCRETE - Assumptions

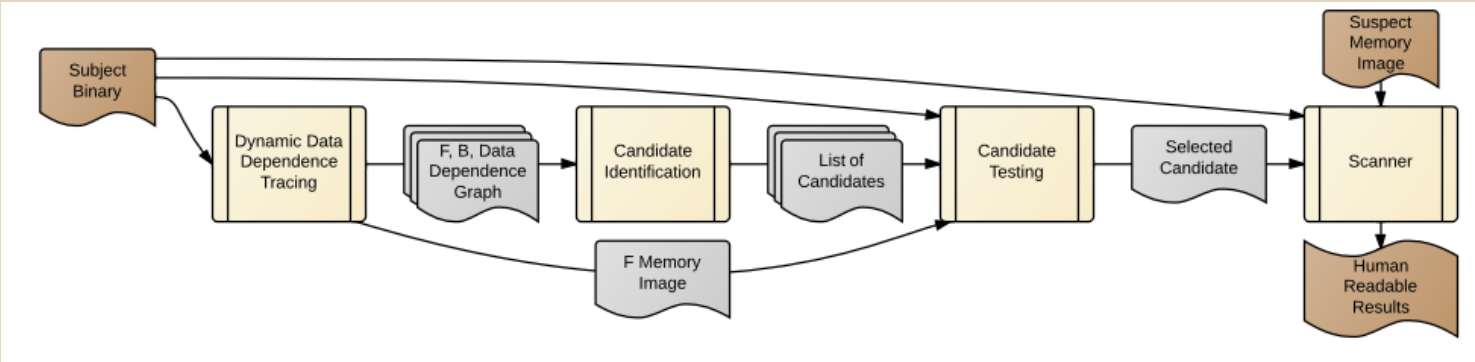
1. The subject binary can be executed (recreate any execution environment)
2. ASLR is disabled

   (Investigator's machine but not suspect's)
3. OS kernel paging data structures in the subject memory image are inact

   (extract memory pages)

DSCRETE - Overview

1. Find P Candidates
   1.1 Tracing; 1.2 Identification;
2. Testing (find real P)
3. Scanner(recover the content)



DSCRETE - Finding candidates

Tracing collects a dynamic data dependence trace from the application binary execution. It will contain the future scanner's code (P!).

HOW: Execute the binary with some inputs (input1) and save the output to a file (output1).

DSCRETE - Finding candidates

DSCRETE will collect each instruction’s data dependency and record lib function or system call invocations as well as their parameters.

Also, DSCRETE saves a snapshot of the stack and heap at the invocation of external lib functions that lead to output system calls.

DSCRETE - Finding candidates

Starting from the system calls that have interesting data structures as parameters, the system finds the closure points.

Closure point:

1. Take interesting data structures as input
2. All selected output/rendering functions must depend on it.

DSCRETE - Testing

Tester: test all the closure points and find P!

HOW: Execute the binary again. Input (input2) is different from input1. When closure point is executed, swap the pointer to input1, see if the output (output2) is valid and same as output1.

DSCRETE - Scanner

Once the P is found, pack P into a scanner tool. And use this tool to recover the content.

P is fed with each offset in suspect’s memory image.

DSCRETE - Example

Consider an example

```
struct pdf* my_pdf;
my_pdf = load_pdf_file(…);
main_loop(my_pdf); // User edits PDF
save_pdf_file(my_pdf);
exit(0)
```

DSCRETE - Example

```
save_pdf_file(struct pdf* ptr)
{
 char* buf = format(ptr);
 fwrite(buf, …);
}
```

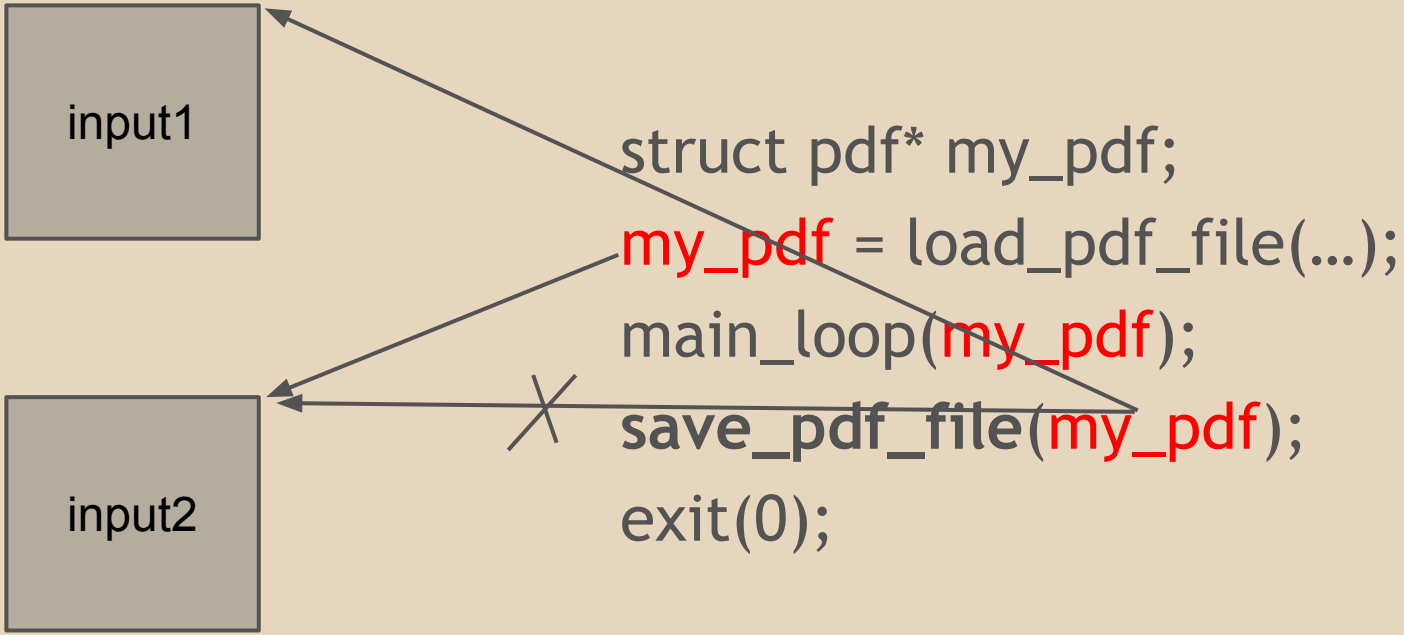
This is the P function! If an invalid input is given, this function will crash.

DSCRETE - Example

During step 1, fwrite will be recorded as well as a snapshot of stack and heap!

The first instruction of **save_pdf_file**(struct pdf* ptr) is the closure point! It loads ptr to argument register and fwrite() depends on it.

DSCRETE - Example



DSCRETE - Evaluation

Almost no FP/FN with reasonable performance

Application	Subject Data Structure	Size (bytes)	True Instances	Total Output	TP	FP	FP%	FN	FN%
CenterIM	`yahoo_data`	160	1	1	1	0	0.0%	0	0.0%
convert	`_Image`	13208	1	1	1	0	0.0%	0	0.0%
darktable	`sqlite3_stmt`	272	1	1	1	0	0.0%	0	0.0%
Firefox	`sqlite3_stmt`	272	1	1	1	0	0.0%	0	0.0%
	`VdbeOp`	24	788	1384	753	502	40%	35	4%
gnome-paint	`GdkPixbuf`	80	51	51	51	0	0.0%	0	0.0%
gnome-screenshot	`ScreenshotApplication`	88	1	1	1	0	0.0%	0	0.0%
gThumb	`GFileInfo`	48	382	381	381	0	0.0%	1	0.4%
	`GdkPixbuf`	80	63	63	63	0	0.0%	0	0.0%
Nginx	`ngx_http_request_t`	1312	6	6	6	0	0.0%	0	0.0%
PDFedit	`XRefWriter`	344	1	1	1	0	0.0%	0	0.0%
top	`proc_t`	720	382	382	382	0	0.0%	0	0.0%
Xfig	`f_compound`	112	1	1	1	0	0.0%	0	0.0%